# Solved

## Catch the Thief [Facebook][M]

Suppose there is a thief and n rooms.

We can only any door to check whether the thief is there or not. during the night, the thief can either move left one room or right one room. Given a sequence of open room.

Check whether we can get the thief using the given sequence.

For example: If there are three rooms, given sequence [1, 1], we can definitely find the thief. if thief was in room[1], we can find him right away. But, if he was in room[0]/room[2], we can find him on the second day since he will have to move one room right/left.

**Solution 1: recursion O(n \* 2#seq)**

在第1步，monster可能在room[0]...room[n-1]，打开门排除掉其中一个room

在第t步，如果t-1步monster在room[k]，则这一步可能在room[k+1]或room[k-1]

这个过程可以画成一棵树，下图是两个n=3时的例子。实线方框标示thief可能在的位置，粗线表示门被打开。

NA

0

1

2

seq[0]=1

1

seq[1]=1

NA

0

1

2

seq[0]=0

1

seq[1]=1

0

2

1

2

seq[1]=2

1

1

左图: input seq = [1, 1] return true.。

右图: input seq = [0,1,2] return false. 虚线表示门被打开但thief不可能在里面。

这样我们得到一个BFS算法。

|  |
| --- |
| **bool can\_catch\_thief(int n, vector<int> seq)** {  vector<int> pre(n);  iota(pre.begin(), pre.end(), 0);  for (int k = 0; k < seq.size() && !pre.empty(); ++k) {  vector<int> cur;  for (int v: pre) {  if (v != seq[k]) {  if (v > 0) cur.push\_back(v-1);  if (v < n-1) cur.push\_back(v+1);  }  }  swap(cur, pre);  }  return pre.empty();  } |

**Solution 2: DP O(n \* #seq)**

上述算法一个显然的优化是可以把pre和cur的容器改成unordered\_set从而把每层的node按照值合并。因为我们只需要判断第t步房间k的存在性，出现一次和出现多次没有区别。这样我们可以把上面的树图化简成下面的样子：

NA

0

1

2

seq[0]=1

1

seq[1]=1

NA

0

1

2

seq[0]=0

1

seq[1]=1

0

2

1

2

seq[1]=2

不难发现我们只需要maintain一个数组记录monster有没有可能在某个房间。（上图中除NA以外的细线方框表示数组中对应位置为1，其它位置为0.）令这个数组为P[][]，不难发现P的第t行只依赖于第t-1行，所以我们只需要维护它的最后两行。

|  |
| --- |
| **bool can\_catch\_thief(int n, vector<int> seq)** {  vector<bool> pre(n, 0);  for (int t = 0; t < seq.size(); ++t) {  vector<bool> cur(n, 0);  int count = 0;  count += (cur[0] = pre[1]) + (cur[n-1] = pre[n-2]);  for (int k = 1; k < n-1; ++k)  count += (cur[k] = (pre[k-1] & pre[k+1]) | (k==seq[t]));  if (count == n) return true;  swap(pre, cur);  }  return false;  } |

如果n<=64，或者语言支持BigInt的位运算，我们可以把P的每一行用一个二进制数编码。这样计算第k行只需要对当前state做几次位运算即可。

|  |
| --- |
| **bool can\_catch\_thief(int n, vector<int> seq)** {  assert(n <= 64);  const uint64\_t target = n==64? 0xFFFFFFFFFFFFFFFF : (1<<n)-1;  uint64\_t stat = 0;  for (int k : seq) {  stat = (((stat << 1) | 1) & ((stat >> 1) | (1<<(n-1)))) | (1 << k);  if (stat == target) return true;  }  return false;  } |

**Solution 3: iteration O(#seq)**

Solution2的inner loop比较浪费，因为其实我们不关心长串的0，只关心Solution1中的cur包括哪些元素。

我们把Solution1的内部for循环拆分成两步：

(1) 第t晚还没开门时monster可能的房间R[t]

(2) 第t晚开门后monster可能的房间R'[t]（就是把R[t]中的seq[t]删掉）

现在我们考虑R[t]在每个晚上发生的演化。初始时R[t]是[0, n)中的所有数。然后R[t]中的奇数项可以从R'[t-1]得到。(只要k+1, k-1中有一个在R'[t-1]中，则k在R[t]中）。

nights

rooms

右图展示了R[t]可能的演化状态，黄色格子表示seq[t]的位置。红蓝色格子表示不存在于R[t]中的房间（即妖怪不可能存在的位置）。颜色用来区分位置index的奇偶性。

由此我们可以猜想R[t]是连续奇数和连续偶数序列的并。这个结论很容易用数学归纳法证明。

l1+2

l2+2

r2-2

r1-2

所以奇偶数序列可以用它们的边界(l1, r1), (l2, r2)表示。

我们只需要在迭代中维护l1, l2, r1, r2的位置。当l1与r1相接且l2与r2相接时，返回true.否则扫完返回false.

|  |
| --- |
| inline void update(int &l, int &r, int s, int left\_bound, int right\_bound, int n) {  bool ext\_l = (s==l+2), ext\_r = (s==r-2);    if (l >= n-2 || (ext\_l && ext\_r)) {//all pos are on  l = right\_bound-2;  r = right\_bound;  }  else {  l = ext\_l? l+1 : max(l-1, left\_bound);  r = ext\_r? r-1 : min(r+1, right\_bound);  }  }  **bool can\_catch\_thief(int n, vector<int> seq)** {  int m1 = n%2?n-1:n-2, m2 = n%2?n-2:n-1;  int l1 = -2, l2 = -1, r1 = m1+2, r2 = m2+2; //sentinel ends  for (int v : seq) {  update(l1, r1, v, -1, m2+2, n);  update(l2, r2, v, -2, m1+2, n);  swap(l1, l2); swap(r1, r2);  if (l1 == r1 - 2 && l2== r2 - 1) return true;  }  return false;  } |

**Followup: 必胜策略?**

受solution3的启发，我们可以想到一个必胜策略[1, 2, 3, …, n-2, n-2, n-3, …, 1] 。

证明：在前n-2步，我们和thief一样每次移动一格。所以我们和thief的之间的距离改变值只可能是{-2,0,2} ，奇偶性不变。类似的在后n-2步距离的奇偶性也不变。在两次打开room[n-2]之间奇偶性改变了。

假设thief初始是奇数：如果我们和thief运动方向相同，距离不变。如果方向不同，因为我们初始在1，thief一定在我们右边，因此距离-2。因此距离是单调非增、且不可能恒定的（因为房间是有限个）。这样我们一定能在前n-2步抓到thief。

假设thief初始是偶数：因为距离一定是奇数，我们无法在前n-2步抓到thief。但到第n步时，我们和thief的距离是偶数，且thief的位置一定和我们相同或在我们左边。这和thief初始是奇数时前n-2步的情况对称。所以后n-2步我们一定能抓到thief.

# 

# 

# 

# 

## Object-Weight Pair [Two-Sigma][H]

让设计一个数据结构，要求可以存储Object-Weight Pair，实现如下几个接口：1） Update；2） Insert；3） Remove；4） GetRandom. 第四个方法是实现的重点。这个GetRandom的方法是随机地返回一个Object，要求概率满足：此object的weight / 所有weight的和。 楼主想的是HashMap的Key存Object，Value存weight，这样可以轻松实现Update，Insert和remove的功能。至于GetRandom这个方法，楼主是用了一个辅助的Array，用来表明每个Object对应的区间，然后用随机数获得某个index，最后看看这个index在哪个区间，然后就返回该对象。但是缺点是：每每更新，插入或者remove掉某个object的时候，这个辅助的array都要重新计算，有没有更好的方法来解决此题？貌似面试官一直在提normalization 和Denormalization。不能理解面试官要的是什么.

Source: <http://www.1point3acres.com/bbs/thread-198541-1-1.html>

**Solution:**

用一个segment tree的结构来存weights。一个queue用来存放树里available的位置。两个unordered\_map用来存segment tree的叶子结点和object之间的双向mapping。

随机采样：

产生一个[1, sum of weights]范围内的数。搜索cumsum(weight)数组的lower\_bound。这可以通过对segment tree作二分搜索得到。

数据添加、删除、更新

维护queue和unordered\_map的操作比较直观。

对segment tree的修改体现在删除时把原来非0的叶子结点置0. 添加时需要检查是否当前capacity已满，如果已满则需要先resize数组。然后update queue指定的叶子位置就好。

|  |
| --- |
| **class rand\_access\_set** {  public:  **rand\_access\_set() : M(128), tree(M\*2, 0)** {  for (int k = 1; k < M; ++k) loc.push(k); //free spot from 1 to M-1  }    **void add(const string &str, int w)** {  if (loc.empty()) resize();  int i = loc.front();  loc.pop();  update\_leaf(i, w);  idx\_to\_obj[i] = str;  obj\_to\_idx[str] = i;  }    **void update(const string &str, int w)** {  int i = obj\_to\_idx[str];  update\_leaf(i, w-tree[M+i]);  }    **void remove(const string &str)** {  int i = obj\_to\_idx[str];  loc.push(i);  update\_leaf(i, -tree[M+i]);  idx\_to\_obj.erase(i);  obj\_to\_idx.erase(str);  }  **string get\_random()** {  Default\_random\_engine eng;  int key = (eng())%tree[1]+1;  int i = sum\_search(key+1);  return idx\_to\_obj[i];  }  private:  void resize() {  tree.resize(M\*4, 0);  for (int base = M; base; base >>= 1) //move current nodes to new locations  rotate(tree.begin()+base, tree.begin()+base\*2, tree.begin()+base\*3);  tree[1]=tree[2]; //new root  for (int k = M; k < M\*2; ++k) loc.push(k);  M<<=1;  }    int sum\_search(int w) {  int i = 1;  do {  i <<= 1;  if (w > tree[i]) w -= tree[i++];  } while (i < M);  return i-M;  }  void update\_leaf(int i, int diff) {  for (int key = M+i; key > 0; key >>= 1) tree[key] += diff;  }  private:  int M; //smallest power of 2 greater or equal than nums.size()+2  vector<int> tree; //tree stored in array, size is M\*2  queue<int> loc; //free spots  unordered\_map<int, string> idx\_to\_obj;  unordered\_map<string, int> obj\_to\_idx;  }; |

## 

## 

## Max Holiday [G][E]

Your company has N sites. Each month you are free to switch to a new site only once within 12hr from your current site. Each site have different holiday schedules. Given a list of each site's holiday schedue and a matrix of traveling time among cities, return max possible holiday you may enjoy in a year.

**Solution:** 为了简化假设输入的traveling distance改为adjacent sites列表。用动态规划计算第m个月呆在第k个城市的最大假期。

|  |
| --- |
| int max\_holiday(const vector<array<int,12>> &num\_holiday,  const vector<vector<int>> &adj\_office) {  int num\_office = num\_holiday.size();    vector<array<int, 12>> partial\_max\_holiday(num\_holiday, array<int, 12>{0});  for (int k = 0; k < num\_office; ++k) {  partial\_max\_holiday[k][0] = num\_holiday[k];  }    int global\_max\_holiday = 0;  for (int m = 1; m < 12; ++m) {  for (int k = 0; k < num\_office; ++k) {  partial\_max\_holiday[k][m] = partial\_max\_holiday[k][m-1] + num\_holiday[k];  for (int j : adj\_office[k]) {  partial\_max\_holiday[k][m] = min(partial\_max\_holiday[k][m],  partial\_max\_holiday[j][m-1] + num\_holiday[k]  );  }  global\_max\_holiday = max(global\_max\_holiday, parital\_max\_holiday[k][m];  }  }  return global\_max\_holiday;  } |

## 

## 

## Split bill [G][E]

N people go out on a trip and cover a portion of the total expense. After back home from the trip, they want to share the expense by money transfer among each other, so that the bill is finally splitted.

Q1a: find a O(n) method to split the bill with less than N transactions.

Q2: what's the lower bound of #transactions (assuming "net" expense are all non-zero)?

Q3: find a method with minimal number of transactions.

Q4: prove that Q2 is NP-complete.

**Solution :**

**Q1-** 假设m是这些人出去玩的平均数，第k个人已付的钱是a\_k。

令b\_k = m-a\_k是第k个人应该多付的钱。

因此一个必胜解是让第k个人付给第k+1个人cumsum (b\_k)数量的钱。注意此时如果cumsum(b\_k)是负的，则transaction 其实是由第k+1个人付给第k个人，如果是零，则该transaction不必要。

|  |
| --- |
| vector<int> compute\_transaction(const vector<int> &expense) {  vector<int> transaction(expense.size(), 0);  Int total;  for (int k = 0; k < expense.size(); ++k) total += expense[k];  for (int k = 0; k < expense.size(); ++k)  transaction[k] = (k>0?transaction[k-1]:0) - expense[k] + (k+1) \* total/expense.size();  return transaction;  } |

**Q2-**因为每个人都需要进账或出帐得到收支平衡，如果把transcation用一个有向图上的边描述的话每个vertex的出度＋入度至少为1. 这样最少要N/2条边（transaction)。

**Q3-**如果没有任何子集和为0，则必须要n-1个transaction。如果有，则可以把集合split into two parts，然后递归地继续分解。所以做法是穷举所有把这群人分成两分sum=0的分割，从而减少1个section，然后对每个递归分割求最大可能的transaction数。（recursion套recursion感觉太暴力了，但是反正这个问题是NPC...）

**Q4-**Q3中找sum为0的非空子集是subset sum problem，而后者是NPC的。所以Q3也是NPC的。

## CamelCase Search[G][M]

比如有一个Set里面有如下单词：FooBar, FootBall, FooToo, FiBa

Input a pattern, return all the words that match the pattern, e.g.

"F" should return "FooBar", "FootBall", "FooToo", "FiBa"

"FB" should return "FooBar", "FootBall", "FiBa"

"FT" should return "FooToo"

"FoBa" should return "FooBar", "FootBall"

"FBa" should renturn "FooBar", "FootBall", "FiBa"

"FiB" should return "FiBa"

|  |
| --- |
| #include <iostream>  #include <cctype>  #include <vector>  #include <string>  using namespace std;  class CamelSearchEng {  struct Node { //trie node  //lazy impl. for children, ideally would only allocate space for alphabetical chars  Node\* children[256] = { nullptr };  bool is\_word = false;    ~Node() {  for (int k = 0; k < 256; ++k) delete children[k];  }  };  Node\* root;  public:  //allocate trie  CamelSearchEng(const vector<string>& words) {  root = new Node;  for (const string& word : words) {  Node\* t = root;  for (char c : word) {  if (t->children[c] == nullptr) t->children[c] = new Node;  t = t->children[c];  }  t->is\_word = true;  }  };    // deallocate trie  ~CamelSearchEng() {  delete root;  };  // search CamelPattern  vector<string> search(const string& pattern) const {  vector<string> result;  string word;  search\_recur(pattern, 0, root, word, result);  return result;  }  void append\_words\_recur(Node\* t, string& word, vector<string>& result) const {  if (t->is\_word) result.push\_back(word);  for (char c = 'a'; c <= 'z'; ++c) {  if (t->children[c]) {  word.push\_back(c);  append\_words\_recur(t->children[c], word, result);  word.pop\_back();  }  }  for (char c = 'A'; c <= 'Z'; ++c) {  if (t->children[c]) {  word.push\_back(c);  append\_words\_recur(t->children[c], word, result);  word.pop\_back();  }  }  }  void search\_recur(const string& pattern, int first, Node\* t, string& word,  vector<string>& result) const {  size\_t word\_len = word.size();  while (first < pattern.size() && islower(pattern[first]) && t) {  t = t->children[pattern[first]];  word.push\_back(pattern[first]);  ++first;  }  if (t == nullptr) {//no match  ; //stop searching  }  else if (first == pattern.size()) { //found a matched prefix  append\_words\_recur(t, word, result); //append all words under t  }  else { //pattern[first] is upper  char c = pattern[first];  for (int ch = 'a'; ch <= 'z'; ++ch) {  if (t->children[ch])  search\_recur(pattern, first, t->children[ch], word, result);  }  if (t->children[c]) {  word.push\_back(c);  search\_recur(pattern, first+1, t->children[c], word, result);  }  }    //restore word for back tracing  word.resize(word\_len);  }  };  void test(const CamelSearchEng& eng, const string& query) {  cout << "query: " << query << endl;  vector<string> result = eng.search(query);  for (const string& str: result) cout << str << " ";  cout << endl << endl;  }  int main() {  vector<string> dictionary{"FooBar", "FootBall", "FooToo", "FiBa"};  CamelSearchEng eng(dictionary);  test(eng, "F");  test(eng, "FB");  test(eng, "FT");  test(eng, "FoBa");  test(eng, "FBa");  test(eng, "FiB");  return 0;  } |

## Ternary Expression[Snapchat][M]

给一个表示ternary expression的表达式，形如A?B:C，表示如果A=true则返回B，否则返回C

A是T(表示true)，或F(表示false)

B, C是T, F，或者是表示ternary expression的子串

写一个evaluate表达式的函数

例如 "T?F?T:F:T?F?T:F:F"表示的是"T?(F?T:F):(T?(F?T:F):F)"

**Solution :**

我们用二叉树表示这个ternary expression的计算过程，可以发现evaluate的过程就是从根节点二分搜索叶子节点的过程(T往蓝色子结点方向走，F往红色子结点方向走）。

所以问题在于如果当前读到A?之后，怎么实现branch操作。首先如果A=true，则只要指向 ?之后从那里开始evaluate即可。如果A=false，则需要跳过左子树。到达右子数。注意到输入字符串相当于前序遍历print，但在访问左子树前插入?，访问右子树前插入:，因此前序遍历中print左子树到print右子树一共print了n个?和n+1个:, n为左子树结点数。因为访问一个tenary expression？永远出现于:之前，因此访问左子树中不可能出现这种情况。

|  |
| --- |
| bool eval\_ternary(const string& str) {  for (int k = 0; k < str.size(); ++k) {  bool val = str[k++]=='T'? true:false;  if (k==str.size()) return val;  if (str[k++] == ':') return val;  else {  if (val==false) {  for (int nq = 0, nc = 0; nq != nc-1 && k < str.size(); ++k) {  if (str[k]=='?') ++nq;  if (str[k]==':') ++nc;  }  }  }  }  throw invalid\_argument;  } |

**Followup:** 条件也是ternary expression怎么办

A?B:C?D:E 会有两种parsing (A?B:C)?D:E 和A?B:(C?D:E)，因此不能允许ternary expression

# 

# 

# Unsolved

## Scheduling with resource allocation [Uber][H]

设计一个Scheduler:

Input:

List of jobs -> each job has: description, time, resource.

k node -> 100 resource per node

Output:List of schedule

**source:** <http://www.1point3acres.com/bbs/thread-198097-1-1.html>

**thoughts:**

看后面的描述似乎不要求最优，只要求合理就好...也许就是吹吹就好了，maybe看看OS的常见策略...

一个可能的方法：考虑贪心＋按job的time和resource分块，

* 把jobs按照时间分为M档, 每个node服务其中的一档
* 对每档的jobs，优先schedule占大块资源的任务，占小块资源的任务可以检察能不能挤到碎片里去.如果当前可选的nodes有多个，优先放等待时间最短、碎片最小的。

**Four Digit-Code Coverage[G][H]**

Q1: Your alphabet contains 10 characters 0-9, how many possible 4-digit code can you make?

Q2: Find lower/upper bound for the length of a sequence whose 4-digit windows cover all possible 4-digit codes.

Q3: Design an algorithm to find this string.

**thoughts:**

**Q1:** 10000

**Q2:**

lower: 10003 (a sequence of length 10003 has 10000 windows)

upper: 40000 (concatenation of all possible windows of length 4)

**Q3:**

贪心比较简单，但不是最优

下面link号称是最优解，待看懂：

<https://instant.1point3acres.com/thread/153807>

# 

# 

# 

# Unimplemented

## Time Traveling Hashtable[Uber][E]

Design a data structure that can store (String key, String value, int timestamp), for example:

insert("key1", "Value1", 1)

lookup("key1") -> "value1"

insert("key1", "value2", 2)

lookup("key1") -> value2

insert("key1", "value3", 4)

lookup("key1") -> "value3"

lookup("key1", 2) -> "value2"

lookup("key1", 3) -> "value2"

lookup("key1", 0) -> nul

**Solution:**

第一层存key -> (time stamp, value)的映射

第二层存按time stamp排序的map结构

## Ranking by comparison [G][MH]

There are n runners in a game. After the game, we receive a collection of news saying "runner i beats runner j". Output the rank of the runners.

Note: there's no guarantee that the news fully cover all runners, or for all of them to be correct (so there could be a loop like i beats j, j beats k, k beats i).

**Solution 1:** Topological sort with ring detection

* If multiple sort are possible, return an arbitrary one
* If ring detected, return empty list

**Solution 2:** Topological sort with greedy resolution of conflicts

* when ring detected, remove 1 or more edges to detroy the ring structure

**Solution 3:** Ranking by random walk (like pagerank)

## Motion Detector[G][MH]

You are in a square shaped room whose top left is (0, 0), and bottom right is (1, 1). You can move to any point (x, y) in the room along a continuous route. x and y values are floats so there's no way to gridify the room.

Under the room there are many motion detectors, each covers a rounded area of a given center and radius. Given a list of motion detectors, return if you can move in the room from (0, 0) to (1, 1) without triggering any detector.

**Solution:** 通过两两检查motion detector的坐标、再进行union-find，可以找到motion detector作用范围的connected components。对每个motion detector，还可以判断它们是否和房间的上下左右边界相交。如果存在一个connected component同时和上下、或左右边界相交，则不能从左上角走到右下角。

## Greencard app[G][MH]

你有很多绿卡的application，有很多stack of them.每个stack有很多application,多到无法把一个stack全放进memory.这些绿卡申请有重复的。要求是de-dup这些申请，并且保持原有绿卡申请的顺序。对于重复的申请，保留最早的申请。

提示：multilevel Hashing

**Solution:**

假设每个stack是一个存着(doc id, app id, timestamp, content)的文件，doc id是唯一的，app id是不唯一的，但同一个app id没有相同time stamp。

定义一个hash function（比如按mod 10)，把submission按照app id map到N个输出文件中的一个。每个输出文件的app id范围保证能被memory hold。这样，我们为每个输出文件维护一个unordered\_map: app id ->( time stamp, doc id)（这里似乎不能用bloom filter，因为不能错删），来判断是否app id是否已经出现过。如果已经出现过且早于当前set里的time stamp，则删除set里的doc，更新map为当前doc。如果当前doc比较新，则删除当前doc。

## Find words in stream of characters [G][EM]

Design a data structure (and its API) which finds all occurrence of list of str in infinite stream of characters

class word\_hist {

public:

read\_char(char c);

unordered\_set<string>& get\_occurrence();

private:

TrieNode \*root;

vector<TrieNode \*> prefix;

unordered\_set<string> result;

}

**Solution1：**令L为字典里的最长单词长度

维护当前位置为结尾的长为1, ...L子串的在trie tree的对应位置

**Solution2：**在trie里存所有单词的reverse，保存长度为L的历史窗口

## Base Station Localization[G][E]

You are surounded by k base stations of known location and the distance between you and the base stations. Estimate your own location.

**Solution1:**假设基站信号是精确的。求基站信号的交。讨论基站个数和解的唯一性。

**Solution2:**假设基站信号不精确，选择两个相交最小的基站，求它们的交确定初始解并通过使用其它基站验证过滤掉一个。然后以它初始化对目标函数gradient descendant。

## Calendar[G][E]

N people have a shared calendar. Return the longest time span that noone is busy.

**Solution 1:**（适用于人多的情况）

首先问繁忙区间是不是整数（比如都是整点到整点）。如果是的话用bit set把有人繁忙的区段标出来，扫一遍就好了。

**Solution 2:**（适用于时间范围很大，人不多的情况）

把所有的繁忙区间merge到一起。merge之后还剩下的缝隙就是所有人不忙的时间段。

## Black List[G][MH]

Return a random number between [0, N) with a blacklist. The blacklist is sorted, and numbers in the blacklist cannot be returned. The blacklist\_random() function should generate a uniform distribution, and minimize the system call rand() as few times as possible.

**Solution:**

假设blacklist排好序。对不在blacklist中的数k，假设blacklist[n]<k<blacklist[n+1]，则rank(k)=k-n.

因此我们randomly产生一个[0, N-#blacklist)之间的数，设为r。

然后我们search for r: 满足 blacklist[n] < r+n < blacklist[n+1]

=> blacklist[n]-(n-1) <= r < blacklist[n]-n 所以我们搜索blacklist[k]-(k-1)的lower\_bound就行

得到n后k = r+n

## Number Game[G][M]

给一个数组，A,B轮流从头尾处选出一个数字，求当B使用（1）贪心和（2）最优策略时，A能取到所有数字之和最大。我使用的recursive写的，优化用的是hash 存储从子数组（i, j）上能得到的最优解。写了几个test跑过了。

**Solution:** 用maxsum[i, j]表示[i, j]范围内以A开始的A最大获益和最小获益。

## Knight on a Chessboard[G][M]

给一个n\*n的chess board，knight可以跳2\*3的格子的对角线，就是国际象棋的规则。求给出一个起始点，knight能否跳到给定的终点。

**solution：**用BFS遍历所有跳法，保存每个结点的上一个位置。最后backtrace。

## Flood in a 2D grid[G][H]

题目就是给你一个matrix，里面的数字代表bar的高度，现在说降雨量如果高于bar的高度水可以漫过去，降雨量0开始每天+1这样，问最早第几天水可以有一条路径从src漫到dst。

**Source:** <http://www.1point3acres.com/bbs/thread-191771-1-1.html>

**Solution:** 从src和dst开始向外扩张，以高度为优先级。当两个分枝联通时的高度为结果。

## Most frequent time stamp[H]

interval [startTime, stoptime) ----integral time stamps

给这样的一串区间 I1, I2......In

找出 一个 time stamp 出现在interval的次数最多。.

startTime <= t< stopTime 代表这个数在区间里面出现过。

example： [1,3), [2, 7), [4, 8), [5, 9)

5和6各出现了三次， 所以答案返回5，6。

**Solution:** 维护一棵segment tree。然后求每个结点的count。结点和结点之间可以share父结点的信息。

## Scheduling with cooldown time [F][H]

给一个task序列ABBABBC, 和相同task的最小interval. 例如interval=3, 则BB运行

时间为5. 写一个函数输入task序列和interval, 输出运行时间。

　followup: 写一个调度函数，输入task序列和interval，输出task最优执行序列

**Source:**<http://www.1point3acres.com/bbs/thread-190631-1-1.html>

**Solution：**和k character apart一样思路。

## Color Conversion[G][T]

Q1: Design a class for RGB colors with integer RGB values.implement operator +.

Q2: Design a method to return whether the RGB is the sum of a subset of given RGB values.

Q3: Design a method to discretize the RGB values. E.g. the allowed values are {0x00, 0x33, 0x66, 0x99}, we get #2f3d13 -> #333300

**Solution**

**Q1:** struct of three int fields r, g, b. + is defined as c1.r+c2.r, c1.b+c2.b, c1.g+c2.g

**Q2:** memorized recursion

**Q3:** (1) break string into the three fields (2) for each field, find nearest neighbor (by binary search)

## Find string rearrangement[G][M]

Q1: Given two strings s1, s2, find the first substring of s1 which is a rearrangement of s2.

Q2: What if s1 is too long to fit into memory?

**Solution：**维护S1当前一个长度为S2的window，以及window的histogram。

不理解为什么原帖说要用divide & conquer

## popular element[G][E]

Find the popular number in the array. Popular means it appeas more than 1/4 size of the array

Q1: if the array has been sorted, can you do it in O(log n) time?

Q2: if the array is not sorted, can you do it in O(n) time?

**Solution:**

**Q1:** 把数组平均分成8份，检查pivot的值。对任意两个pivot，如果它们的值一样为v，则可能在对应的3/8段子序列里有一个长>¼数组长度，值为v的小段。此时只要二分搜索找出最后一个<v的数和第一个>v的数就好了。

**Q2:** popular element最多有4个。如果我们把array里的数字每4个不同的放一组的话，剩下的数字就有可能是popular element（也可能不是）。

## Typesetting[G][M]

You have an empty page modeled as a 2D matrix of R rows and C columns. Your goal is to put a text into the matrix. The text consititutes of n words separated by space ' '.

Q1: if you get to put the text as many times as possible, how many times can you fill the text completely? Can you make it O(RC)?

Q2: you pay a cost of x^2 for not filling x spaces in each non-empty. Design an algorithm to typeset the text with minimal cost.

|YOU HAVE AN |

|EMPTY PAGE YOU|

|HAVE AN EMPTY |

|PAGE YOU HAVE |

|AN EMPTY PAGE |

**Solution:**

Q1: 肯定是要尽量塞满每一行，这个结论可以反证（把没有塞满的最优塞法的相应单词放到上一行即可。

所以我们需要知道对所有的k, 从第k个单词开始要继续放多少个单词可以塞满一行.假设这个单词是n，n可能大于单词总数。这通过对字符串做sliding window可以计算。

然后用O(r)时间，我们可以计算每行有几个单词。

Q2:(DP) 维护前k行放了n个词的最小cost。

## Validate Circular Matrix[G][M]

Check if a matrix is a circular matrix, e.g.

1 2 3 4

5 1 2 3

6 5 1 2.

7 6 5 1

with the following constraints

(1) the memory can fit a single row

(2) the memory can fit a portion of a single row

**Solution:** 按照题意矩阵是存在文件里。因此我们维护两个读文件的iterator，第一个比第二个快C+1个数字，C为列数。同时我们维护第一个指针所在列数。

这样circular matrix的validation变成：

* 从文件中读两个数
* 如果当前第一个iterator指向最后一列，continue
* 否则判断两个iterator所值内容是否不同，如果不同返回false

直到第二个iterator读完所有元素，返回true

对本题提及的两种可能性，可以按照内存大小批量把文件内容load到cache里。

## Buy and sell stock [G][M]

Say you have an array for which the ith element is the price of a given stock on day i. Design an algorithm to find the maximum profit. You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Q1. you may complete as many as possible transactions, but at least 1 transactions.

Q2. you may complete as many as possible transactions, but each of transaction requires paying a cost of c.

Solution:

Q1: 只要检查股价是不是一直在跌。如果不是，则返回没有1 transaction条件的解。否则返回最小的隔天差价。

Q2: maintain 最后一个transaction在第k天结束的最大收益local，和transaction在第k天以前结束的最大受益global。local的更新incorporate transaction fee即可。

## Meeting Room III [Snapchat][M]

meeting room II的变体，不是求需要多少个room，而是要把schedule打出来，输出哪些meeting在同一个room里进行。很轻松就写完了，然后学长说，那我们变一下，现在只有一个room，然后每个meeting都有权重，要求在不冲突的情况下使得最后安排下来的权重和最大

**source:** <http://www.1point3acres.com/bbs/thread-172729-1-1.html>

Solution:

Q1: 简单修改Meeting Room II就好了。

Q2: （DP）计算在time t以前满足会议要求的最大权重和。如果t是整数，还比较简单。如果t是浮点数，可以考虑把会议按结束时间排序。然后把dp结果存在一个map里。

## String Compression[G][MH]

We can compress a string of alphabetic characters into a (possibly shorter) format using [] to brace repeativie parts. e.g. abcaaaaa -> abc5[a].

Q1: Given a compressed string, examine if it is valid.

Q2: Uncompress a compressed string. What is the time complexity?

Q3: Find shortest compression of an alphabetc string

**Solution:** 需要clarify一些边界条件。如数字串可不可以0开始，可不可以就是0.

**Q1-** 类似检查parenthesis。保证左括号数大于等于右括号数。且结束时需相等。这里左括号必须由数字串开始。

**Q2-** 用一个栈压[之前的字符串，看到]的时候把内容append到[前面的字符串就好了

需要注意问一下比如5b[a]可不可以，还是数字一定表示个数，以及数字应该不会溢出吧...

**Q3-**

假设最短压缩的形式是

<q0>n0[p0]<q1>n1[p1]....<qk-1>nk-1[pk-1]<qk>

其中<q>里没有循环节，但可以是空串

n[p]对应一个top-level的循环节:

* p非空，可以是形如<q0>n0[p0]<q1>n1[p1]....<qk-1>nk-1[pk-1]<qk> 的压缩串
* n是循环次数，n>1
* len(p)=1时，要求n>＝4 (aaaa 和4[a]一样长）
* len(p)=2时，要求n>＝3(abab比2[ab]短，ababab比3[ab]长）
* len(p)>=3时，要求n>＝2(abcabc和2[abc]一样长）

所以可以递归遍历输入串的所有top level分割，对循环节部分递归调用最短压缩得到该分割下的最优压缩。最后输出所有分割方式中最优压缩最短的。

有三个部分可以利用memorization优化：

* top level分割中倒数k个字符的最优分割，因此可以把对分割的recursion转化为一个DP
* 子串p的最优压缩
* Cache以每个字符开始的长为l的子串的最多循环次数，这样对n0[p0]...形式的结构不用重复检查n0次p0，只要检察前两个p0是否match就好

感觉面试可能聊到这就好了，甚至把前面的递归说出来就可以...

进一步优化: 预存所有循环子串的"原子"top level压缩

* proposition:（原子循环节存在）令一个循环串的最短循环节长度是p，它的任意其它循环节长度q是p的整数倍  
  反证假如互质，则q的首字母出现在串的0, 2q, 3q...位置，对应p中的0， 2q%p, 3q%p...位置，这覆盖了p的所有位置。这意味着串的最短循环节是1。
* 推论: 一个循环串s的最优压缩n[p]满足p是s的最短循环节，且p不是循环串  
  第二个结论的意思是m[n[a]]不如(m\*n)[a]

伪代码：

function comp(str)

如果str在cache中，直接返回

for k = len(str)-1:-1:1

//求suffix = str长为k的后缀的最优压缩

comp(suffix) = suffix[0]+comp(suffix[1…])

for l = 1: k

if suffix长为l子串存在原子压缩n[p], n>1：

cp = comp(p); min\_m 为把cp作为循环节需要的最少重复次数

for m = min\_m : n //这里也许还有优化空间  
 comp\_candidate = n[comp(p)] comp(suffix[l…])

如果comp\_candidate更短， 更新comp(suffix)=comp\_candidate

cache and return comp(str)

它的复杂度应该少于计算所有子串的最优压缩，对后者只需要考虑内层循环，所有递归调用都是O(1)时间的，对长度为k的子串，复杂度< k + k/2 + k/3 + … + 1 = O(k log k)。

所以整个算法的复杂度 sum\_k (n-k) \* k \* log(k) 大约是O(n3log n), 如果最里层能优化成常数则可能可以改进到O(n3)

找循环子串及其原子循环节：O(n3)

for k = 1: len(str)/2

for s = 0: k-1

检察s + nk 开始的子串是否有长为k的循环子串

## Number of Islands III [G][H]

Q1: Implement a class modeling a 2D binary image supporting the following ops

1. Set/Unset a point
2. Return #islands (connected components) in the image

Q2: can you extend this to 3D?

Q3: find rank 2 islands (大岛内湖的湖心岛）

<http://www.1point3acres.com/bbs/thread-197882-1-1.html>

**Solution**

Q1: Set用union find。检察Unset的点需要检察该点是不是cut vertex，google了一下似乎必须DFS。

Q2: 3D和2D没有区别，只是领域增加到6个点

Q3: 首先label 0的联通分量。rank 1 lake: 和边界相交的lake。rank 1 island：和边界或rank 1 lake相交的island。rank 2 lake: 和rank 1 island相交的lake, rank 2 island：和rank 1 lake相交的island

# 

# Snapchat面经

## Basics

### 1. 二维数组对角线打印 ［T]

1 2 3 4

5 6 7 8

9 10 11 12

打印出

1

2 5

3 6 9

4 7 10

8 11

12

|  |
| --- |
| **void print\_matrix\_diagonal(const vector<vector<int>>& matrix)** {  if (matrix.size() == 0 || matrix[0].size() == 0) return;  for (int col = 0; col < matrix[0].size(); ++col) {  for (int r = 0, c = col; r < matrix.size() && c >= 0; ++r, --c) {  cout << matrix[r][c] << " ";  }  cout << endl;  }  for (int row = 1; row < matrix.size(); ++row) {  for (int r = row, c = matrix[0].size()-1; r < matrix.size() && c >= 0; ++r, --c) {  cout << matrix[r][c] << " ";  }  cout << endl;  }  }  // test  **int main()** {  cout << "case 1:" << endl; // square matrix  vector<vector<int>> mat1 {{1, 2}, {3, 4}};  print\_matrix\_diagonal(mat1);  cout << "case 2:" << endl; // wide matrix  vector<vector<int>> mat2 {{1, 2, 3}, {4, 5, 6}};  print\_matrix\_diagonal(mat2);  cout << "case 3:" << endl; // tall matrix  vector<vector<int>> mat3 {{1, 2}, {3, 4}, {5, 6}};  print\_matrix\_diagonal(mat3);  cout << "case 4:" << endl; // empty matrix  vector<vector<int>> mat4 {};  print\_matrix\_diagonal(mat4);  return 0;  } |

### 2. anagram substr [E]

给定字符串A，B， 判断A中是否存在子串是B的anagram

|  |
| --- |
| **bool substr\_anagram(const string& str, const string& pattern)** {  int hist[26] = {0};  for (char c : pattern) ++hist[c-'a'];    int matched = (1 << 26)-1;  for (int k = 0; k < str.size(); ++k) {  int i = str[k] - 'a';  if (--hist[i] == 0) {  matched &= ~(1 << i);  }  else {  matched |= (1 << i);  }    if (k >= pattern.size()) {  int j = str[k - pattern.size()]-'a';  if (++ hist[j] == 0) {  matched &= ~(1 << j);  }  else {  matched |= (1 << j);  }  }  if (matched == 0) return true;  }  return false;  } |

### 3. Encode/decode message［T]

Given a message, implement encode() and decode() such that:

**encode()**

E.g. message is “hello world”

(1) return distinct chars -> 8

(2) sort chars by frequency -> “lohe wrd”

(3) translate the message with the following dictionary: “l” -> 01 “o” -> 011 “h” -> 0111 “e” -> 01111 … So in the example, after translation it would be “0111011110101….” for this function return (1)+(2)+(3) as a string, i.e. “8lohe wrd0111011110101….”

**decode()** take in the output from encode, return the message string.

**FOLLOW UP QUESTION:** for encode, how to save space? e.g. (3) translation, we can store them as bits then convert every 8 bits as a char to return.

<https://instant.1point3acres.com/thread/188946>

**Solution:** 就是先用01串前面的东西建字典，然后parse后面的0-1串，感觉很trivial...

### 4. Big Int[M]

Big integer addition and substraction, multiplication.

string add(string s1, string s2){

}

big number add

两个很大的数相加，要考虑小数.

实现两个整数相除。

**Solution:** 目测没有什么算法，但是实现需要练一下

|  |
| --- |
| int cmp\_positive(const string& lhs, const string& rhs) {  if (lhs.size() > rhs.size()) return 1;  if (rhs.size() > lhs.size()) return -1;  return lhs.compare(rhs);  }  string add\_positive(const string& lhs, const string& rhs) {  string result;  int carry = 0;  for (int k = 0; k < lhs.size() || k < rhs.size(); ++k) {  int vl = k<lhs.size()?(\*(lhs.rbegin()+k))-'0':0;  int vr = k<rhs.size()?(\*(rhs.rbegin()+k))-'0':0;  int sum = vl + vr + carry;  result.push\_back(sum%10+'0');  carry = sum/10;  }  if (carry) result.push\_back(carry+'0');  reverse(result.begin(), result.end());  return result;  }  string minus\_positive(const string& lhs, const string& rhs) {  auto cmp = cmp\_positive(lhs, rhs);  if (cmp < 0) return "-"+minus\_positive(rhs, lhs);  if (cmp == 0) return "0";  string result;  int borrowed = 0;  for (int k = 0; k < lhs.size(); ++k) {  int vl = \*(lhs.rbegin()+k);  int vr = k<rhs.size()?(\*(rhs.rbegin()+k)):'0';  int diff = vl - vr + borrowed;  if (diff < 0) {  borrowed = -1;  diff += 10;  }  else borrowed = 0;  result.push\_back(diff + '0');  }  int len = result.find\_last\_not\_of('0');  if (len == string::npos) return "0";  else result.resize(len + 1);  reverse(result.begin(), result.end());  return result;  }  class big\_int {  private:  string val;  public:  big\_int(): val("0") {};  explicit big\_int(const string& val\_) : val(val\_) {};  big\_int& operator+=(const big\_int& rhs) {  int code = (val[0] == '-') << 1 | (rhs.val[0] == '-');    switch (code) {  case 0: //both positive  val = add\_positive(val, rhs.val); break;  case 1: //rhs negative  val = minus\_positive(val, rhs.val.substr(1)); break;  case 2:  val = minus\_positive(rhs.val, val.substr(1)); break;  case 3:  val = "-"+add\_positive(val.substr(1), rhs.val.substr(1)); break;  }  return \*this;  }    big\_int& operator-=(const big\_int& rhs) {  big\_int minus\_rhs(rhs.val[0]=='-'?rhs.val.substr(1):"-"+rhs.val);  (\*this) += minus\_rhs;  return \*this;  }    friend big\_int operator+(big\_int bi1, const big\_int& bi2) {  bi1 += bi2;  return bi1;  }    friend big\_int operator-(big\_int bi1, const big\_int& bi2) {  bi1 -= bi2;  return bi1;  }  friend ostream& operator<<(ostream& os, const big\_int& bi) {  os << bi.val;  return os;  }  }; |

### 5. 用Array实现ArrayList[E]

**Solution:** 相当于问vector自动allocate空间的部分

|  |
| --- |
| class vector\_by\_array {  int capacity = 64; //initial capacity  int size = 0;  int \*arr;    void resize() {  int \*new\_arr = new int[capacity\*2];  copy(arr, arr+size, new\_arr);  delete[] arr;  arr= new\_arr;  capacity \*= 2;  }    public:  vector\_by\_array() { arr = new int[capacity]; }  ~vector\_by\_array() { delete[] arr; }  void push\_back(int number) {  if (size == capacity-1) resize();  arr[size++] = number;  }  void pop\_back(int number) {  if (size == 0) throw "empty array";  arr[size--] = number;  }  }; |

### 6. find even split[T]

给一个数组，返回左边之和等于右边之和的那个index， 没有就返回-1，比如{1,2,3,2,1}， 返回2

**Solution:**从左到右扫一遍，算sum。再扫一遍时，维护前缀和并检查后缀和是不是等于前缀和就好了。

|  |
| --- |
| int find\_even\_split(const vector<int>& numbers) {  int sum = 0;  for (int n : numbers) sum += n;  int psum = 0, k = 0; //prefix sum  for (int n : numbers) {  if (psum \* 2 + n == sum) return k;  psum += n;  ++k;  }  return -1;  } |

### 7. Pattern match[E]

第一轮，ABC小哥，给两个string的数组和一个pattern数组，判断将两个string数组分别和pattern转化后的结果是否相同。例如s1={“abc”, “a”, “ccc”}, s2={“bc”, “aa”, “d”}, pattern={1, 0}，则s1和p的转化结果是“aabc”，s2和p的转化结果也是是“aabc”，则返回true。如果pattern是{0, 1}，则转化结果分别是“abca”, “bcaa”，则返回false。followup是，如果给定s1和s2，判断是否存在一个pattern使得转化结果一致。 过程中要求分析算法复杂度。

s1，s2，s 分别代表string list1，string list2 和index list。两者根据index list，match就return true，不match就return False。我一上来先提出了直接历变的解法，集创建两个空string，根据index不断把两个string list里的string往两个空string里面写。小哥问我时间复杂度，我说O(n)，他说太general了，具体咧？懵逼了，真没了解那么深。后来提醒我说每次空string叠加也是耗时的，耗的时间跟被叠加的string有关。。。就是O(mn)。再问我空间复杂度，我说O(n)，还说general，最后也是O(mn)，让我改进我的算法用O(1)的空间复杂度。。。好吧，在不断交流和提醒中，写完了，用pointer，四个pointer，两个指现在的string，两个string里面的字母。但是时间也到了，我明白后面还有follow up，我没做成，第一轮扑街。

**Solution:**

### 8. Implement diff/ Word Splitter. [EM]

**Solution:**

### 9. String sort with number cmp [E]

是关于给一组string sort，规则是除了正常的按ascII字母排序，纯数字的比较需要看完整数字。这里有一些细节需要搞清楚，比如overflow以及首位为0的情况特殊处理

**Solution:**

## Stack

### 1. parse function log[E]

function\_name start\_or\_end timestamp

f1 start 1

f2 start 3

f2 end 5

f1 end 7

f3 start 9

f3 end 10

返回一些interval表示函数的运行interval,如上面这个log就返回

f1 [1,3] [5,7]  
f2 [3,5]  
f3 [9,10]

<https://instant.1point3acres.com/thread/188907>

**Solution:** 这里需要问一下自己调用自己的情况要不要把连续的item merge起来，如果需要的话再result.emplace\_back那里加一个判断并且在要插入的name和back()的name相等时修改back的end time就行。

|  |
| --- |
| struct function\_info {  string name;  unsigned int start;  unsigned int end;    function\_info(const string& name\_, unsigned int start\_, unsigned int end\_)  : name(name\_), start(start\_), end(end\_) {};  };  struct function\_event {  string name;  bool is\_start;  unsigned int time\_stamp;    function\_event(const string& name\_, bool is\_start\_, unsigned int time\_stamp\_)  : name(name\_), is\_start(is\_start\_), time\_stamp(time\_stamp\_) {};  };  vector<function\_info> parse\_log(const vector<function\_event>& log){  if (log.size() == 0) return vector<function\_info>{};    vector<function\_info> result;  stack<string> stk;  int start = log[0].time\_stamp;  for (const auto& e : log) {  if (!result.empty()) result.emplace\_back(stk.top(), start, e.time\_stamp);  if (e.is\_start) stk.push(e.name);  else stk.pop();    start = e.time\_stamp;  }    return result;  } |

### 2. Basic calculator[EM]

（2\*3） ＋ 6 这种，还有大括号，小括号等要处理

<https://instant.1point3acres.com/thread/184629>

+-\*/和数字 都是一位 没有空格 输入都是valid 所以不用考虑corner case

follow up1: 数字变成多位

follow up2: 加上空格

follow up3: 负数

上面两个问题

**Solution:** 标准做法是先转后缀表达式然后再evaluate

|  |
| --- |
| inline void update\_operands(stack<int>& operand\_stk, stack<char>& operator\_stk) {  char op = operator\_stk.top(); operator\_stk.pop();  int operand2 = operand\_stk.top(); operand\_stk.pop();  int operand1 = operand\_stk.top(); operand\_stk.pop();  switch (op) {  case '+': operand\_stk.push(operand1 + operand2); break;  case '-': operand\_stk.push(operand1 - operand2); break;  case '\*': operand\_stk.push(operand1 \* operand2); break;  case '/': operand\_stk.push(operand1 / operand2); break;  }  }  int shunting\_yard(const string& expression) {  istringstream iss(expression);  stack<char> operator\_stk;  stack<int> operand\_stk;    int level[256] = {0};  level['('] = level[')'] = 1; level['+'] = level['-'] = 2; level['\*'] = level['/'] = 3;    char lastc = '(';  while (true) {  char c = iss.peek();  if (iss.eof()) break;  if ((!isdigit(lastc) && lastc != ')' && c == '-') || isdigit(c)) {  int num;  iss >> num;  operand\_stk.push(num);  }  else {  iss.ignore(1);  if (c == ' ') ;  else if (c == '(') operator\_stk.push(c);  else if (c == ')') {  while (!operator\_stk.empty() && operator\_stk.top() != '(')  update\_operands(operand\_stk, operator\_stk);  operator\_stk.pop();  }  else {  while (!operator\_stk.empty() && level[operator\_stk.top()] >= level[c])  update\_operands(operand\_stk, operator\_stk);  operator\_stk.push(c);  }  }  lastc=c;  };  while (!operator\_stk.empty()) update\_operands(operand\_stk, operator\_stk);  return operand\_stk.top();  }  int main() {  cout << shunting\_yard("-2 +( -13/ -5)") << endl;  cout << shunting\_yard("2+ 3\* (4+6) /5") << endl;  return 0;  } |

## Hashing

### Toggle snap[E]

给我看了一个snapchat给好友群发消息的功能，可以任意选中和删除想要投递消息的好友，并显示群发好友的list(按先后选中的顺序)，设计一个数据结构，实现 toggle(String username); getList();

**Solution:**hashmap + doubly linked list的设计, 类似LRU cache

## Graph

### 1. Undirected graph representation

题目是设计一个类来存放一个无向图的边和点，follow up是问图有多满的情况下用bitmap更加合适。

<http://www.1point3acres.com/bbs/thread-167175-1-1.html>

**Solution:** 这个就是比较adjacent list v.s. 2D boolean matrix的表示

vector of vector占的空间大概是V+E的数量级，但是每个entry都是32位int，加上每个vector本身可能有些数据占常数)空间假设是c。则adj list的空间至少要32(V+E) + c(V+1) bit， bitmap的空间是V^2 bits，甚至可能减半。根据比较两者可以算出来一个E的上界，超过这个上界就是bitmap更好。

另外一种情况是两个node之间有多条边。此时bitmap的每个entry需要用一个unsigned int存，参数上会略有变化。但分析方法一样。

### 2. Topological sort

给了一个class如下

class Task {

public:

void exec() {};

set<Task\*> getDeps() { }

};

然后写一个函数，输入是一堆Task，每个task可能有一些dependency，必须先执行完dependency才能执行这个task，输出是这堆task的正确执行顺序。

follow up是如果有环，需要报出错误信息。同时可以补充给的这个class。依然是用递归写的。。。

<http://www.1point3acres.com/bbs/thread-167175-1-1.html>

**Solution:** 还是topological sort，用BFS/DFS都行

### 3. Bipartrate graph［E]

opposite gender check. Given a list of user pairs, return whether or not it's possible all the pairs are of opposite gender. E.g. oppositeGender() should return false because if (A, B) is of opposite gender, (B, C) is of opposite gender, then (A, C) can NOT of opposite gender. 我的solution就是建个tree/ graph，然后BFS treverse.

<https://instant.1point3acres.com/thread/188946>

<https://instant.1point3acres.com/thread/189105>

**Solution:** BFS, 顺便为每个node maintain一下label

吐槽这个题政治不正确..

|  |
| --- |
| vector<int> color\_bipartite(int nvertex, const vector<pair<int, int>>& edge) {  vector<int> label(nvertex, -1);  vector<vector<int>> adj(nvertex, vector<int>{});    for (const auto& e : edge) {  adj[e.first].push\_back(e.second);  adj[e.second].push\_back(e.first);  }    for (int head = 0; head < nvertex; ++head) {  if (label[head] >= 0) continue;  label[head] = 0;  queue<int> q;  nodes.push\_back(head);  while (!q.empty()) {  int src = q.top();  q.pop();  for (int dst : adj[src]) {  if (label[dst] == label[src]) {  return vector<int>{};  }  if (label[dst] < 0) {  label[dst] = 1 - label[src];  q.push(dst);  }  }  }  }  return label;  } |

### 4. Inorder traversal of DAG[M]

给一个DAG, 求出进行inorder traversal后的第k个node。此DAG上每个node out degrees最大为2.但是in degree可以大于2。这个题目我当时没做出来。因为一个点可以被visit好多次。

<https://instant.1point3acres.com/thread/143720>

**Solution:** 最坏情况left and right node都指向同一个node，导致这个node会被重新in-order traverse一遍。

所以假如n个node的最坏时间是T(n)，则有T(n)=2T(n-1)+1, T(1)=1，可以计算出通项公示T(n) = 2n-1

求第k个node的写法就是正常的中序遍历。

### 5. Max score in k steps[E]

给了一个Person class， 有score和nextSnap两个属性，每个Person有一个朋友的list，也就是nextSnap，输入是（Person， maxStep）， 在maxStep步数以内算max score，注意的是下一个Friend可能会指向上一个Person，要注意回溯的问题。。。

**Solution:** BFS走maxStep层

|  |
| --- |
| int max\_score(Person\* query, int maxStep) {  unordered\_set<Person\*> visited;  int score = INT\_MIN;  vector<Person\*> pre(1, query);  visited.insert(query);  for (int k = 0; k < maxStep && pre.empty(); ++k) {  vector<Person\*> cur;  for (Person\* p : pre) {  score = max(score, p->score);  for (Person\* snap : p->nextSnap) {  if (visited.find(snap) != visited.end()) continue;  visited.insert(snap);  cur.push\_back(snap);  }  }  swap(pre, cur);  }  return score;  } |

### 6. Max score in k steps II [EM]

snap with weights, snap has next snaps list

input是start snap, max steps求出一个max total weights

follow up: start snap是一个list怎么办

follow up: 如果每个snap都可以是start nap怎么办， memorization search

BFS的方法做就可以.

**Solution:** BFS

Followup1 : 添加一个(virtual)src node，然后max step +1

Followup2:

### 7. Group redundant users [E]

手机上的通讯录，每条记录只有(name, number)这种pair,有些记录名字重复，有些记录号码重复，让我返回一个list<list<Record>>，将所有记录按人分组。比较tricky的点在于(ABC,123), (ABC, 456), (BCD, 456)三条记录，第一条和第三条也属于同一个人。要求时间复杂度尽量小。

**Solution:** union find

### 8. Find common friends [E]

给定一个有向关系图，在给定2个名字，求出这两个人的共同朋友（即2个点能达到的共同节点，类似树的共同祖先）

## 

## Priority queue, heap

### 1. Object collision[MH]

说很多object移动 速度方向 都不一样 overlap的话都消失 怎么检测某个时刻 哪些object还在 这一轮很多条件都要是和大哥讨论得出来的 他并没有给很多条件 最后自己设计object的class 然后写出来了 大哥挑了一个bug 改了之后就好了

<https://instant.1point3acres.com/thread/189105>

**Solution:** 题目没有说object形状，暂且假设它们是点。如果是圆的话就是求二次方程，会麻烦比较多。

暂时只想到一个简单的解法。对任意两个物体，求它们轨迹的交点，从而知道(1)它们会不会相交 (2)如果会，相交的时刻。然后对所有可能相交的pair，按照时间排序。

### 2. Circle overlap[M]

给你很多圆，用圆心和半径表示，求有没有圆是相交的。<https://instant.1point3acres.com/thread/184976>

**Solution:** 首先用line sweeping可以判断圆的外接矩形是不是相交。然后对所有外接矩形相交的圆可以判断是不是真的相交。

### 3. Drive through Radar[M]

coding 题目是有一条路，路两边可以想象为 y = 0 和y = 1两条直线。现在给你list of radar，每个雷达为(横坐标，纵坐标，辐射半径)。问你一辆车能否通过这条路。

**Solution:** line sweep

### 4. task management design[MH]

比方说给你50个task，有个API假设已经有了，是用来run这些task的，但是这个API最多每次只能同时run3个。并且run的顺序根据priority level来定，让实现。具体包括 get，change priority 和run，get就是新的task，change priority就是把已有的还没运行的task更改priority level，run就是运行同时保证运行完一个马上推入下一个运行，

**Solution:** thread pool with priority queue?

## Tree

### 1. Bad IP filtering[E]

Implement storeBadIps(): storing a list of bad ips, and isBadIp(): return whether or not a given ip is a bad ip.  
  
\* storeBadIps()  
E.g. storeBadIps(["7.0.0.0/8", "10.3.0.0/16", "6.7.8.134/32"]).  
In the example "7.0.0.0/8", "7.0.0.0" is ip ranges from 0.0.0.0 to 255.255.255.255, "8" is how many bits from the beginning it needs to match. in this example any "7.x.x.x" would be a bad ip.  
In the example "10.3.0.0/16", any "10.3.x.x" would be a bad ip.  
In the example "6.7.8.134/32", only exact match "6.7.8.134" would be a bad ip.  
  
\* isBadIp()  
E.g. isBadIp("7.3.4.5") should return true. isBadIp("6.7.8.0") should return false.

<https://instant.1point3acres.com/thread/188946>

<https://instant.1point3acres.com/thread/184976>

**Solution:** 不知道storeBadIps是不是只call一遍。感觉搞个class比较靠谱。

总之store的过程中首先把ip转成32进制数，然后存在一个binary tree结构的trie里。然后isBadIp的时候搜这棵树。

|  |
| --- |
| class bad\_ip\_lookup {  struct trie\_node{  bool is\_bad = false;  trie\_node\* children[2] = {nullptr, nullptr};    ~trie\_node() {  delete children[0];  delete children[1];  }  };    trie\_node\* root;  int encode\_ip(const string& ip) {  int first = 0;  int code = 0;  for (int k = 0; k < 4; ++k) {  int num = 0;  for (; isdigit(ip[first]); ++first) {  num = num\*10 + (ip[first]-'0');  }  code = code << (8 + num);  ++first; //skip . or /  }  if (first < ip.size()) {  int nbit = 0;  for (; isdigit(ip[first]); ++first) {  nbit = nbit\*10 + (ip[first]-'0');  }  code = code >> (32-nbit);  }  return code;  }  public:  bad\_ip\_lookup() {  root = new trie\_node;  }    ~bad\_ip\_lookup() {  delete root;  }    int store\_bad\_ip(const string& ip) {  int code = encode\_ip(ip);  int b = 31;  for (; b > 0 && ((code>>b)&1)==0; --b);  trie\_node\* node = root;  for (; b >= 0; --b) {  int id = (code>>b)&1;  if (node->children[id] == nullptr) node->children[id] = new trie\_node();  node = node->children[id];  }  node->is\_bad = true;  }    bool is\_bad\_ip(const string& ip) {  int code = encode\_ip(ip);  for (; b > 0 && ((code>>b)&1)==0; --b);  trie\_node\* node = root;  for (; b >= 0; --b) {  int id = (code>>b)&1;  if (node->children[id] == nullptr) return false;  node = node->children[id];  }  return node->is\_bad;  }  }; |

### 2. Tab completion[EM]

第一轮电面：实现 linux command line中的tab completion功能（其实就是让实现一个trie，但是要注意一下有多个可能结果时的情况。小哥当时要求实现的效果跟command line behavior一样。<https://instant.1point3acres.com/thread/153520>

**Solution:** 真的就是建trie，然后把trie下面的children全部前序print一遍...

|  |
| --- |
| class trie {  struct trie\_node {  trie\_node\* children[26];  bool is\_word = false;  ~trie\_node {  for (int k = 0; k < 26; ++k) delete children[k];  }  };    trie\_node\* root;  void add\_node(const& string prefix) {  trie\_node\* node = root;  for (char c : prefix) {  if (node->children[c] == nullptr) return node->children[c] = new trie\_node;  node = node->children[c];  }  node->is\_word = true;  }    trie\_node\* find\_node(const& string prefix) {  trie\_node\* node = root;  for (char c : prefix) {  if (node->children[c] == nullptr) return nullptr;  node = node->children[c];  }  return node;  }    void find\_children\_recur(trie\_node\* node, string& prefix, vector<string>& result) {  prefix.push\_back(' ');  if (node->is\_word) result.push\_back(prefix);  for (int k = 0; k < 26; ++ k) {  if (node->children[k] == nullptr) continue;  prefix.back() = 'a'+k;  find\_children\_recur(node->children[k], prefix, result);  }  prefix.pop\_back();  }  public:  trie(vector<string> dict) {  root = new trie\_node();  for (auto& word : dict) add\_node(word);  }    ~trie() {  delete root;  }    vector<string> auto\_complete(const& string prefix) {  vector<string> result;  trie\_node\* node = find\_node(prefix);  if (node == nullptr) return;  vector<string> result;  string prefix;  find\_children\_recur(node, prefix, result);  return result;  }  }; |

### 3. word abbreviation[M]

题意： 我们通常压缩的时候会把 interval 压缩成 i8l, 即首尾字母加这个word的长度。

1. 但是研究人员发现， internal 和 interval 如果按上面那种方法就会模糊不清，因为都会压缩成 i8l. 于是研究人员就想到一个办法，就是加长它们的prefix, 直到遇到它们第一个不同的字母， 比如：internal 和 interval 会压缩成： intern8l, interv8l. intension 和 intrusion 会变成： inte9n, intr9n

2. 当压缩完后， 发现压缩后的长度 大于等于 原始长度时， 保留原始长度。比如 intern8l 长度也是 8， 那么就 保留原始： internal. (例子我自己举的， 大概这意思) input: 是一个 string like god internal me internet interval intension face intrusion output: l2e god internal me i8t interval inte9n f2e intr9n (注意： 输出的时候要按照 输入string 的顺序， 顺序不能变)。

<http://www.1point3acres.com/bbs/thread-162312-4-1.html>

**Solution:** 用一个trie确定每个单词的前缀至少要跳到哪，然后第二遍扫的时候就可以完成压缩

需要按照长度和尾字母分别建trie

|  |
| --- |
| class trie {  struct trie\_node {  trie\_node\* children[26];  bool is\_word = false;  int count = 0;    ~trie\_node() {  for (int k = 0; k < 26; ++k) delete children[k];  }  };  trie\_node\* root = nullptr;    public:  trie() { root = new trie\_node; }  ~trie() { delete root; }    void add\_word(const string& str) {  trie\_node\* t = root;  for (char c : str) {  int id = c-'a';  ++t->count;  if (t->children[id]==nullptr) t->children[id] = new trie\_node;  t = t->children[id];  }  ++t->count;  t->is\_word = true;  }    int find\_prefix(const string& str) {  int len = 0;  for (char c : str) {  if (len > 0 && t->count == 1) break;  int id = c-'a';  if (t->children[id]==nullptr) return -1;  t = t->children[id];  ++len;  }  return -1; //entire word is prefix of others  }  };  vector<string> word\_abbreviation(vector<string> original) {  unordered\_set<string> history;  trie tree;  vector<string> compressed;  for (const auto& str : original) {  if (history.find(str) == history.end()) {  tree.add\_word(str);  history.insert(str);  }  }  for (const auto& str: original) {  int prefix\_len = str.find\_prefix(str);  string str\_len = to\_string(str.size()-2);  if (str.size() <= 2 || prefix\_len + str\_len.size() + 1 >= str.size() ) {  compressed.push\_back(str);  }  else {  compressed.push\_back(str.substr(0, prefix\_len) + str\_len + str.back());  }  }  return compressed;  } |

### 4. Implement map with BST[E]

以BST实现map，只需要支持int型hash到int型。 几个接口函数：

int getValue(int)

void insert(int, int)

bool erase(int)

<https://instant.1point3acres.com/thread/185226>

**Solution:** 就是BST的node存key, value，检索用key。如果要求平衡就上treap...需要练一下erase

|  |
| --- |
| #include <iostream>  using namespace std;  class bst\_map {  struct node {  int key;  int val;  node\* left;  node\* right;  node(int key\_, int val\_) : key(key\_), val(val\_), left(nullptr), right(nullptr) {}  ~node() { delete left; delete right;}  };  node\* root = nullptr;    static node\*& search(int key, node\*& root) {  if (!root || key == root->key) return root;  else if (key < root->key) return search(key, root->left);  else return search(key, root->right);  }    static bool is\_leaf(node\* n) {  return n->left == nullptr && n->right == nullptr;  }  public:  bst\_map() {}  ~bst\_map() { delete root; }    bool insert(int key, int val) {  node\*& found = search(key, root);  if (found) return false;  found = new node(key, val);  return true;  }    bool erase(int key) {  node\*& found = search(key, root);  if (found == nullptr) return false;    if (is\_leaf(found)) {  delete found;  found == nullptr;  return true;  }  bool go\_left = (found->left != nullptr);  auto lchild = [=](node\* n)->node\* { return go\_left? n->left : n->right; };  auto rchild = [=](node\* n)->node\* { return go\_left? n->right : n->left; };    node\* parent = found;  node\* child = lchild(found);  while (rchild(child) != nullptr) {  parent = child;  child = rchild(child);  }    found->val = child->val;  found->key = child->key;  if (child->key < parent->key) parent->left = lchild(child);  if (child->key > parent->key) parent->right = lchild(child);  return true;  }    int get\_value(int key) {  node\* found = search(key, root);  if (found == nullptr) return 0;  return found->val;  }  };  int main() {  bst\_map bm;  bm.insert(1, 3);  cout << bm.get\_value(1) << endl;  bm.insert(2, 4);  bm.insert(3, 5);    cout << bm.get\_value(2) << endl;  cout << bm.get\_value(3) << endl;  bm.erase(2);  cout << bm.get\_value(1) << endl;  cout << bm.get\_value(2) << endl;  cout << bm.get\_value(3) << endl;  } |

### 5. closest point[M]

我们有一个data sets, 然后有一个graph是类似10 \* 10的grid,写一个function, input是一个query point，求出离它最近的点.

brute force是每个点distance求一下，后来改成了类似quad tree分成四个area的做法.

给定一个grid matrix，就是类似围棋盘那种东西。然后某些grid（放棋子的地方）上面有点，给定一个query点的位置，返回k nearest点on this grid matrix。第一开始我assume给了个list，里面是自己设计的一个结构体，记录了坐标和距离query点的距离，然后写了个comparator来sort这个list，然后返回k个node。之后要求优化到O(n)，楼主就写了quick selection。之后要求再优化，然后楼主就\*\*了，尼玛这是要让我写kd-tree的节奏，

### 6. XML parser[E]

coding题目是XML parser，follow up是一开始我们假设xml是well-formatted,如果不是的话，比如有不成对出现的<p>或者</p>

XML parser, build a map to represent XML given tokenizer.Next() -> (open,name), (inner,content), (close,name), 就是一个 tree transversal

不过有一点不同是需要设计data structure保存paser后的结果。 每个token有如下结构，然后给了个API getNextToken()获取下一个token, 我表示应该还有个

hasNextToken()。

and

token {

string name; // e.g. story, id, snaps

string tag\_type; // {open, close, test} 三种type

};

### 7. TXT to tree[E]

题目是input一个text file，里面是last name, first name, mm/dd/yy, manager's full name这样的格式，按照他的要求parse然后按照hierarchy print出来。

Q1. print 结构就是一般的indentation

Q2. 第一个child前面跟 |- 最后一个前面是\- ，其余是|

### 8. design query suggestion system[E]

implement trie tree's seach function.

### 

## Algorithms

### 1. The maze problem[E]

第二轮还是个中国小哥，题目有点类似wall and gate，就是一个二维迷宫，0表示可以走，1表示障碍物，问从其中某点到另一点有没有通路。用BFS+染色法做了。follow up是打印出这条路（不必是最短路径）是什么<http://www.1point3acres.com/bbs/thread-167175-1-1.html>

问题一个走迷宫问题。给了一个矩阵，"1"代表起点，位于左上角；"9"代表终点，位于右下角；"0"代表通路，"5"代表墙。

矩阵长得是这样

[

[1, 5, 5, 5, 5, 0],

[0, 5, 0, 5, 5, 0],

[0, 5, 0, 0, 0, 0],

[0, 5, 0, 0, 5, 0],

[0, 0, 0, 5, 0, 9]\

].

设计一个算法，看看这个迷宫能不能从起点走到终点。

<http://www.1point3acres.com/bbs/thread-178130-1-1.html>

**Solution:** DFS+mask

让你从左边col的任意点走到右边col的任意点的最小step。走不到就返回-1  
注意起始点是任意左col的点 而不是左上 ， 也不是走到右下。 要考虑到中间如果有一个col全部都是0（墙）的case

<http://www.1point3acres.com/bbs/thread-198749-1-1.html>

**Solution:** BFS。

如果起点可以第一列任意一个点就加一个src node就行。

coding题目是一个矩阵里某些位置有墙，给一个出发点及目的地，求最短距离。 follwup是现在墙可以打破，没打破一个cost为1，求cost最小的路线。

**Solution:** 还是BFS，只是如果有墙的话，需要delay到下一个layer

### 2. Simple words[EM]

given an array of words. output array of simple words. compound word are made of multiple simple words. chat ever snapchat snap salesperson per person sales son whatsoever what so 其中simple word是chat, ever, snap, per, sales, son, what, so

<https://instant.1point3acres.com/thread/161003>

**Solution:** 把单词按照从短到长排列。每进来一个新单词，对它作word break。如果成功，则该单词不是simple word。如果失败，则单词是simple word，输出并把该单词假如字典。

### 3. Combination sum with negative numbers[E]

给一个array比如,target = 4. 返回有多少种组合使得和为4.可以重复取, 无顺序. 好像是lc combination sum IV. 很快做完. 有负数怎么办: 不能有组合使得和为0. follow up. 可以有负数，增加一个参数表示可以做最取几个负数. 简单的dfs. 不是很难. 也比较快做完..

<https://instant.1point3acres.com/thread/188907>

**Solution:** 有负数的版本多传一个变量表示还允许搞几个负数就可以了。

### 4. number removal[E]

给一个数组，元素是0-9的int，remove掉其中n个数，使剩下的数在原顺序的情况下最小。e.g. 7129，去掉两个数字，最小为12。 57294，去掉两个数字，剩下的最小为294。复杂度 O(length + n)。 follow up: 如果其中有一个负数怎么办？这里你需要跟面试官clarify怎么处理负数。我clarify的结果是，负数不能在剩下数的中间，例如1 -2 9，是invalid的，负数只能在剩下数的首位。例如-2 1 3。例如：7 -1 3 4，去掉一个，剩下的最小为-134。8 6 1 -8 6，去掉两个，因为-8前面有三个数，所以必须把-8去掉。

<https://instant.1point3acres.com/thread/184976>

**Solution:** 第一问似乎就是leetcode上字母序最小的那个，用一个stack记录结果，尽量让当前结构单调增

第二问：如果有负数，则负数后面如果位数足够，一定要从负数开头，且选负数最小的那个中最小的。后面的选字母序最大的，跳过负数。

### 5. Amiable pair[E]

给一个数n，返回所有amicable pair，要求pair中第一个数小于第二个数，然后问了问复杂度。

### 6. Ads placement[M]

一段视频中有一些可以放广告的位置，长度不同，有一堆可以选的广告，长度也不同，问怎么放最多和最长的广告

## DP

### 1. Number picking game[M]

A跟B在play game，从int[]里拿数字 A：拿第一个/最后一个，都有可能 B：greedy，总拿第一个/最后一个中最大的 A先开始，然后B，轮流，直到拿完 求A拿到的最大 举个例子： {3, 7, 2, 1} A: {3, 1} -- 4 B: {7, 2} -- 9 A: {1, 7} -- 8 B: {3, 2} -- 5 答案是8

<https://instant.1point3acres.com/thread/175275>

**Solution:** DP每一段对应的最大分数。

**Followup:** 两个人从1-N里面不重复地取数出来加在同一个sum上，第一个超过target的人赢，求先手是不是能赢。 写的暴力recursive

### 2. Number of ways followup[E]

Given a M X N grid, a random cell in the grid (coordinate for example), find number of ways you can reach top-left corner in k steps, assuming you can move in any direction.

<https://instant.1point3acres.com/thread/153520>

**Solution:** DP到达每个grid长度为k有几条路。

### 3. Palindrome edit distance[E]

给一个string,可以在任意位置add/delete/change，还可以combine几种来使他变为合法的palindrome。比如：ebabc可以先删了中间的a再把c改成e。问最少需要几步。

**Solution:** DP每个子串str[i...j]的最小edit distance

str[i...j]分几种情况：

- str[j]后面插入str[i]，代价为C[i+1, j]+1，删除str[i]代价一样

- 类似地，str[i]前面插入str[j]，代价为C[i, j-1]+1，删除str[j]代价一样

- 修改str[i]以match str[j]，代价为C[i+1, j-1]+1

终结条件:

i==j或i>j: C[i, j] = 0

### 4. Party organization[E]

题目是你要举办一个party，给你一些人和他们的上下级关系，如果你邀请了一个人就不能邀请他的直接上级或者直接下级，问怎样邀请能使邀请的所有人的级别加起来和最高（比如CEO是10级然后VP 9级这样，类似每个人有一个分数）。

<http://www.1point3acres.com/bbs/thread-167175-1-1.html>

**Solution:** 假设graph是个tree，后序遍历并更新每个node下面的包括root的最大value和不包括root的最大value

Inclusive\_max[node] = max\_{c in child of node} exclusive\_max[c] + score[node]

exclusive\_max[node] = max\_{c in child of node} inclusive\_max[c]

## System design

### 1. Debugging web app/What happend after typing an URL

然后问如果有一个web based的email app，打开的latency很高，应该怎么办，如何test哪个部分，如果Improve.

如果有准备过what happened after type an URL，因为没有太大问题，无非就是按照那个过程，DNS, proxy server, cache(这是重点)

**Solution:**

1. browser checks cache; if requested object is in cache and is fresh, skip to #9
2. browser asks OS for server's IP address
3. OS makes a DNS lookup and replies the IP address to the browser
4. browser opens a TCP connection to server (this step is much more complex with HTTPS)
5. browser sends the HTTP request through TCP connection
6. browser receives HTTP response and may close the TCP connection, or reuse it for another request
7. browser checks if the response is a redirect or a conditional response (3xx result status codes), authorization request (401), error (4xx and 5xx), etc.; these are handled differently from normal responses (2xx)
8. if cacheable, response is stored in cache
9. browser decodes response (e.g. if it's gzipped)
10. browser determines what to do with response (e.g. is it a HTML page, is it an image, is it a sound clip?)
11. browser renders response, or offers a download dialog for unrecognized types

### 2. implement bloomfilter

construct, add, mightcontain, delete, extend(if near full, extend)

**solution:** confused... bloomfilter doesn't support delete I think, and not sure what is extend

### 3. Server allocation

要求设计和实现一个server分配的功能，如果一个server被使用了，返回下一个可用的server。

<http://www.1point3acres.com/bbs/thread-198032-1-1.html>

**Solution：** lz扯了一些load balancing

感觉要讨论的东西比较多，比如是否有sticky session的问题，server是否可能被动态加减，server有没有使用上限。感觉扯一些consistent hashing的东西也可以？

算法上可以扯maintain一个可用的服务器队列，blahblah

### 4. Design twitter

然后一上来就出了一道twitter系统的设计，幸亏我之前准备过system design，而且在其他面经上也看到别人提到过这个，然后给他讲了讲各个方面的设计。 比如消息推送的机制，push和poll。还有怎么样用hybrid方法。

### 5. Design google doc

设计一个比较简单的Google doc。如何保证大家拿到的 信息是一致的，在同一个version.

### 6. Design snapchat

对视频进行处理比如变形，加各种表情装饰等，问怎么实现这个系统，这轮不用coding，纯design

### 7. Design buffer management

实现一个buffer管理系统，不断的存入图片，图片大小不同，然后另一边不断的读出图片，类似于实现一个blocking queue, 但是要求底层实现，连list都不能用

### 8. Design Snapchat Story feature

有private，public用户，24hr ttl 删除之前的story

### 9. Design Snapchat Discover feature

### 10. Rate limiter

class Throttler {

int qps;

public Throttler(int qps) {

}

pubilc boolean allowAccess() {

}

}

给这么一个class实现rate limiter，allowAccess()返回的事当前时间的access能不能被批准

举个栗子：

qps是2

request1 time 0.0 return true;

request2 time 0.5 return true;

request3 time 0.6 return false;

### 11. Find common friends in social network

在一个很大的social network中，给两个个名字，如何快速的找到名字对应的node然后再去找他们的common friends。楼主提了一下hash，consistent hash然后就没啥了，不熟悉DHT没办法啊。还是自己准备不够充分，好多东西需要了解。

### 12. Evaluate friend recommendation on a large social network

然后问说我们有个很huge的social network，你需要design一个system，这个system可以用来evaluate各种我们设计的朋友推荐算法，你要怎么设计。说了一下大概的设计，面试官更看重我如何准备data来做evalution，于是就也说了说这个部分。

## OO design

### 1. Rectangle drawing

问题是假定给个屏幕和一些屏幕坐标和渲染单个像素点的API，要求设计一个渲染class, 能实现当手指点下去的时候开始渲染，手指拖动过程中渲染以屏幕一角和手指作为对角线的一个长方形（这个长方形可以根据手指的运动扩大缩小），最终当手指离开屏幕的时候，以手指最后位置为准，留下most recent的那个长方形。<https://instant.1point3acres.com/thread/153520>

**Solution：**当前状态和上一个状态，可能是在画，或者不在画

finger click事件发生的时候需要记录起点位置

finger down过程中需要刷新painting，比较慢的做法是把之前的像素set off，然后set on当前像素

快一些的做法是如果两次终点在同一象限的话，求上一次渲染的长方形和这次渲染的长方形的差。code会复杂一点。

finger up时状态改为不在画，把上一个状态时画的所有pixel set off

### 2. Rotate screen

给手机的画图app写个屏幕旋转的method。同样问了许多design问题。

### 3. design four in a row game/m x n tic tac toe

两个人是在一个m\*n的board上玩。规则如下：

（1）获胜方式依然是横竖对角线有三个连在一起的symbol。

（2）每次movement不能任意放置，只能放在 每一列 的 最下方的空白处，也就是说每个玩家每轮最多只有 n （行数）个选择。

要求实现以下API：

（1）valid()。检查当前board是否有效，有效board必须满足

(i) 没有人获胜

(ii) 不能违背第二条规则。

（2）nextMove()。返回当前玩家的任意一个movement，

要求对手无法获胜，如果找不到报错（我选择了返回-1）

脑子实在累了，讨论了一下给出了bruteforce的方案。

1. valid主要难点在怎么判断已经有人赢了，LZ用check8个方向一共16个格子的方法，于是O(16\\*N^2)。跟小哥交涉，表示常数可以减小，不过16也合理可以写。

2. 假设当前为player A， 则枚举A的n个选择，每个选择会对应B的n个选择，复杂度依然是O(16\*N^2)。但实际上如果不保存board state，不管是A还是B都得先找到每一列放置的位置，每次都扫描一遍就会多出O(N)的时间，总时间会变成O(N^3)，所以需要保存一下状态。

## Multithreading

### 1. Sort on disk

最后还有时间就问了一个如何用4G内存sort 100G数据，假设CPU速度很快可以忽略，瓶颈是硬盘读写速度（50M/S），问只有一块硬盘的话估计要用多少时间，如果有两块硬盘如何优化。

<http://www.1point3acres.com/bbs/thread-167175-1-1.html>

**Solution:** 目测是外排序。

### 2. 多线程版本queue using stack

followup: 自己实现mutex那些，自己写测试，问了我如何设计测试等等。

怎样test保证能在product work，扯了下unit test + modular test + system test, 然后具体说了unit test的五六种case，然后test发现有个小bug，改之。

### 3. 单机服务器handle大量request

让我coding一个单机的web 服务器的消息处理机制。 我先说如果是单机的话，就不用考虑scale的问题。说了好多乱七八糟的system design的问题，但是他不是这个意思。后来搞了两三分钟才讲清楚是要实现一个类。说当访问量大的时候，怎么handle request，我说线程池。 他说yes， that's what i want。 然后其实就是实现线程池类。

### 4. Write heavy counter

写一个可以被多线程访问的计数器？？每被访问一次count++，check被访问次数的method并不常用

## Machine learning

### 1. Derive back propagation

### 2. Derive kernel trick